Scheduling Algorithms for Peer-to-Peer Collaborative File Distribution

Jonathan S.K. Chan Victor O.K. Li King-Shan Lui
Department of Electrical and Electronic Engineering
University of Hong Kong, Pokfulam, Hong Kong, China
{skjchan, vli, kslui}@eee.hku.hk

Abstract

Peer-to-Peer file sharing applications on the Internet, such as BitTorrent, Gnutella, etc., have been immensely popular. Prior research mainly focuses on peer and content discovery, overlay topology formation, fairness and incentive issues, etc, but seldom investigates the data distribution problem which is also a core component of any file sharing application. In this paper, we present the first effort in addressing this collaborative file distribution problem and formally define the scheduling problem in a simplified context. We suggest several types of algorithms, including a novel Bipartite Matching algorithm, for solving the problem. Simulation results show that our weighted bipartite algorithm finds an optimal solution for all cases tested. Therefore, we believe our algorithm is a promising solution to be employed as the core scheduling module in P2P file sharing applications, shortening the total download time experienced by users.

1. Introduction

Peer-to-Peer (P2P) applications have become immensely popular in the Internet. Among P2P applications, collaborative sharing of large video/audio files and software is perhaps the most popular one. Compared with traditional client/server file sharing approaches (e.g. FTP, WWW), P2P collaborative file sharing has one major advantage, namely, scalability. The performance of client/server approach deteriorates rapidly as the number of simultaneous clients increases, since the outgoing bandwidth of the server is shared among all simultaneous clients. Interestingly, in a well-designed P2P file sharing network, more peers participating in the file sharing session generally means better performance, as each peer could download simultaneously from multiple peers without any obvious bottleneck links.

Experiments, such as those in [1], have shown that using parallel downloading scheme in P2P file sharing systems, in which an end user opens multiple connections with multiple file sources to download different portions of the file from different sources and then reassembles the file locally, could result in higher aggregate download rate and thus shorter download time. Due to the significant performance improvements with collaborative file sharing, there has been widespread use of P2P file sharing applications like BitTorrent [2], Gnutella [3], Kazaa [4], Napster [5], etc., just to name a few.

P2P file sharing systems have been attracting much research attention for the past few years since its recent inception. Although there are many papers related to P2P systems investigating issues of diversified interest, most of them focus on topics like overlay topology formation, peer discovery, content search, sharing fairness and incentive mechanisms, etc. Few of them really look into the data distribution scheduling problem, which is the core of any file sharing systems since without actual data transmission and distribution, no file sharing is possible. A schedule tells each peer which piece should be sent and to whom. A poor data distribution schedule could result in considerably longer download time, while a good schedule could shorten the completion time and efficiently utilize all resources like network bandwidth. This article presents the first effort in addressing the "data distribution scheduling problem" in P2P collaborative file sharing systems and proposes a novel bipartite graph model for efficiently solving the scheduling problem.

The major contributions of this paper are summarized as follows:

- We propose the push-based decision model instead of the commonly used pull-based model, in order to eliminate the transmission of file piece request messages.
- We formally define the data distribution scheduling problem and derive a lower bound of transmission time for a simplified scenario.
- We propose several algorithms (including a novel bipartite graph model) for determining the file piece distribution schedule, and evaluate their performance by simulations.
- Simulation results show that the bipartite graph model finds the optimal schedules for all the cases we tested.

The rest of this paper is organized as follows. Section 2 briefly describes some related work. The communication

1-4244-0030-9/05/$20.00 ©2005 IEEE

model, notations and analysis are given in Section 3. Section 4 presents various algorithms for approaching the scheduling problem defined in Section 3, followed by an evaluation of simulation results in Section 5. Some of our future directions are presented in Section 6 and we conclude the paper in Section 7.

2. Related Work

Due to the dramatic increase of broadband user population, there has been large-scale deployment of P2P file sharing systems in the Internet. BitTorrent [2, 6] is one of the most popular P2P file sharing applications with thousands of simultaneous users. A shared file is chopped into multiple small pieces (each about 256Kbytes or 512Kbytes). Some tracker servers are used to periodically announce the list of connected peers who participate in the same sharing session. Each peer then uses this peer list to contact other peers and reports to them which pieces he currently possesses, and requests those missing pieces he does not have from those peers who have them. A peer can maximize its downloading speed by requesting different pieces from different peers at the same time. A poor scheduling algorithm may lead to every peer getting nearly the same set of pieces and consequently decreases the number of file piece sources which a peer can simultaneously download from. BitTorrent employs the *Rarest Element First* algorithm, in which those pieces that most peers do not have are downloaded first. This algorithm is good at increasing the availability of different file pieces in the network and can distribute all pieces from the original source to different peers across the network as quickly as possible. However, our simulation results show that *Rarest Element First* is not an optimal scheduling algorithm.

From the algorithmic point of view, a series of papers [7, 8, 9, 10] has investigated the problem of broadcasting or multicasting a single message in heterogeneous networks. [7] studies the performance on completion time of various algorithms in a network where nodes have different processing times and the transmission times between different node pairs are also different. It shows that the well-known *Fastest Node First* may result in solutions which are worse than the optimal by an unbounded factor and subsequently proposes the *Fastest Edge First* and *Earliest Completing Edge First* algorithms to better solve the problem. If all transmission times are the same but nodes have different processing times, [8] proves that the problem of minimizing the maximum completion time of broadcasting a single message in networks is NP-hard. It also shows the *Fastest Node First* heuristic in computing broadcast schedules can produce an 1.5 approximation schedule for the same problem. However, it should be stressed that the above papers [7, 8, 9, 10] only analyze the case when there is only one single message for transmission. Our aim in this paper is to analyze the case when the "message" is chopped into multiple pieces situated at different nodes waiting for complete distribution. It is obvious that our problem is much more complex than previous work and we present the first effort in addressing this problem based on some simplifications.

3. Problem Definition

Due to the initial complexity we face when first studying this problem, we develop our model based on several simplifications and assumptions.

3.1. Communication Model

3.1.1. Homogeneous Network. We assume a homogeneous network model to begin our first attack to the problem. In a homogeneous network, all peers have the same uploading and downloading bandwidth. The transmission time for sending a message from any node to any other node is the same. An example is shown in Figure 1, where all peers are situated at the edge of the network, with logical links connecting every pair of peers (i.e. fully-connected graph).

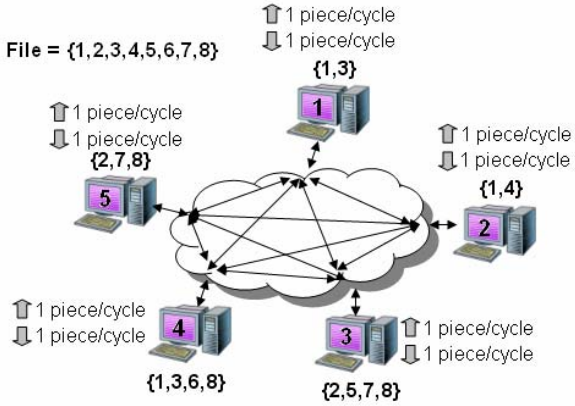


Figure 1. Homogeneous network

3.1.2. Notations and Definitions. Let the number of participating peers be N and the number of file pieces be M. The shared file F is chopped into M small pieces $F = \{F_1, F_2, \ldots, F_M\}$ and each peer possesses a subset of F. We represent the file piece possession information as an $N \times M$ matrix P, called the *possession matrix*. $P_{ij} = 1$ if and only if node i possesses file piece F_j ($1 \le i \le N$, $1 \le j \le M$); otherwise $P_{ij} = 0$. We use P^t to denote the possession matrix at time t. Refer to Figure 1 where $F = \{F_1, F_2, \ldots, F_8\}$ and $\{\ldots\}$ next to a node indicates the pieces that the node possesses, the possession matrix at the beginning (t=0) is:

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Due to the homogeneous network assumption in Section 3.1.1, the data distribution can be made in discrete cycles synchronously. In each cycle, each peer can only send and receive at most one file piece. Given an initial possession matrix $\boldsymbol{P^0}$, after one cycle of file piece distribution, a new possession matrix $\boldsymbol{P^1}$ will be formed. That is, $\boldsymbol{P^k}$ denotes the possession matrix after $\boldsymbol{k}$ cycles.

We also refer to a possession matrix as a *problem instance* since it provides all the information we need to solve the problem. A problem instance $\boldsymbol{P}$ is *feasible* if for each file piece in $\boldsymbol{\{F_1, F_2, \ldots, F_M\}}$, at least one peer possesses it. That is, $\sum_{i=1}^{N} \boldsymbol{P_{ij}} \geq \boldsymbol{1}, \forall \boldsymbol{j} \in \boldsymbol{[1, M]}$. A problem instance is *infeasible* if it is not feasible, meaning that there is no way for every peer to get all file pieces since there is at least one file piece not available in the system.

A *schedule* specifies how file pieces are distributed among peers. At each cycle, for each peer, a schedule determines which file piece the peer has to send out and to whom. A possible schedule for $\boldsymbol{P^0}$ above is:

Node 1: send piece 3 to node 2
Node 2: send piece 4 to node 1
Node 3: send piece 5 to node 5
Node 4: send piece 6 to node 3
Node 5: send piece 2 to node 4

Formally, we use two matrices to represent the schedule in one cycle. We use an $\boldsymbol{N \times M}$ matrix $\boldsymbol{T}$ to represent the *piece transmission* schedule, which specifies which piece a peer receives. $\boldsymbol{T_{ij} = 1}$ if and only if node $\boldsymbol{i}$ receives file piece $\boldsymbol{j}$, otherwise $\boldsymbol{T_{ij} = 0}$. We use another $\boldsymbol{N \times N}$ matrix $\boldsymbol{S}$ to indicate the senders of the file pieces. $\boldsymbol{S_{ij} = 1}$ if and only if node $\boldsymbol{i}$ receives file piece from node $\boldsymbol{j}$; otherwise $\boldsymbol{S_{ij} = 0}$. Similar to $\boldsymbol{P}$, we use superscript to refer to different cycles. That is, $\boldsymbol{T^k}$ and $\boldsymbol{S^k}$ together form the schedule to be performed at cycle k. For the above example schedule, we have:

$$S^0 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad T^0 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Node 1 receives piece 4 sent from Node 2 and therefore, $\boldsymbol{S^0_{12}=1}$ and $\boldsymbol{T^0_{14}=1}$.

There are several properties that a *valid* schedule (a schedule that does not violate any assumptions) should observe and they are listed as follows. (Superscripts in matrices are dropped to enhance readability when the context is clear.)

- There must be at least one file piece distributed among the peers in each cycle (at least one entry in $\boldsymbol{S}$ and $\boldsymbol{T}$ is $\boldsymbol{1}$)
- A node cannot send a piece that it does not possess (if $\boldsymbol{S_{ij} = 1}$ and $\boldsymbol{T_{ik} = 1}$, then $\boldsymbol{P_{jk} = 1}$)
- A node cannot send more than one file piece in a cycle (sum of every column in $\boldsymbol{S}$ is at most 1)
- A node cannot receive more than 1 file piece in a cycle (sum of every row in $\boldsymbol{S}$ is at most 1)

It is not difficult to see that a valid schedule may not be a good one. For example, we should not arrange Node 2 to send file piece 1 to Node 1 since Node 1 already possesses that piece. Formally, in a good schedule, $\boldsymbol{T^k_{ij} = 0}$ if $\boldsymbol{P^k_{ij} = 1}$ for the same $\boldsymbol{i, j}$ at any cycle $\boldsymbol{k}$.

Given the possession matrix $\boldsymbol{P^{k-1}}$ and a valid and good schedule $\boldsymbol{S^{k-1}}$ and $\boldsymbol{T^{k-1}}$ at cycle $\boldsymbol{k-1}$, the possession matrix at cycle $\boldsymbol{k}$ ($\boldsymbol{k > 0}$) can be obtained by adding $\boldsymbol{P^{k-1}}$ and $\boldsymbol{T^{k-1}}$. Mathematically, $\boldsymbol{P^k = P^{k-1} + T^{k-1}}$. Refer to the above example,

$$P^1 = P^0 + T^0 = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

Intuitively, if we keep on applying a valid and good schedule in each cycle to a feasible problem instance, all peers will get all the file pieces eventually and the file distribution can terminate. In other words, given an initial feasible $\boldsymbol{P^0}$ and a valid and good schedule, after certain, say $\boldsymbol{k_0}$, cycles, $\boldsymbol{P^{k_0}_{ij} = 1, \forall i, j}$. $\boldsymbol{k_0}$ is the time needed for complete distribution of the whole file to all peers. An optimal schedule is a schedule that requires the minimum number of cycles to complete among all possible schedules. Our goal is to develop algorithms that aim at finding optimal schedules.

3.1.3. Pull-based vs. Push-based. Here we distinguish two models for determining the transmission schedule: *pull-based* and *push-based*. In both models, peers have to exchange their file piece possession information periodically. They differ in how to make the decisions of which piece to send and to whom.

The pull-based model is commonly used in existing applications, such as BitTorrent, in which the receiver determines which file pieces he needs from others and subsequently sends request messages to the nodes he chooses. The file source who receives these request messages could choose to accept or reject the requests based on some policies, such as his available bandwidth and the requestors' contributions. One disadvantage of this model is that there will be many short-length but frequent request messages flowing through the network, taking up network bandwidth and processing time. In addition, for the distributed pull-based model, it may happen that all peers decide to request the same file piece from the same source, thus wasting queuing time at the

source node (or even getting rejected by the source node). For example, in the following problem instance, all nodes may send requests to node 1 for piece 1 since piece 1 is the rarest piece (most needed piece, since only one node in the network has it). We refer to this problem as *request collision*.

$$P=\begin{pmatrix}1&0&1&0&1&1&0&0\\0&1&0&1&1&0&1&0\\0&1&1&0&1&0&1&1\\0&0&1&1&0&1&0&1\\0&1&0&1&0&1&1&1\end{pmatrix}$$

In view of the above problem, we propose the push-based model, in which the sender decides which file piece and to whom he would like to send. In a homogeneous network, in the beginning of a cycle, every node can construct the possession matrix $\boldsymbol{P}$ based on the possession information from its peers and this matrix is the same for every node. Based on $\boldsymbol{P}$, each peer determines the file piece to send and the recipient directly, without the need of request messages used in the pull-based approach. To avoid the request collision problem, an algorithm that generates valid schedules (as defined in Section 3.1.2) is employed. The scheduling algorithm determines what each peer should do based on the possession matrix. As long as each peer executes the same valid scheduling algorithm using the same matrix, peers should send file pieces without any conflict.

In Section 4, we describe several scheduling algorithms that are suitable for the push-based model and evaluate their performance by simulations in Section 5.

3.2. Analysis

In this section, we analyze the lower bound of k_0, which is the number of cycles needed for complete distribution of the whole file to all peers. We consider the case that each peer can both send one file piece and receive one file piece for each synchronous cycle and the problem instance is *feasible* (definition in Section 3.1.2).

Let r_i be the total number of 0s across row i, that is $r_i=\sum_{j=1}^{M}(1-P_{ij})$, we can find the maximum number of 0s across all rows $r_{max}=\max_{i\in[1,N]}\{r_i\}$. Let c_j be the total number of 1s along column j, that is $c_j=\sum_{i=1}^{N}P_{ij}$, we can find the minimum number of 1s along all columns $c_{min}=\min_{j\in[1,M]}\{c_j\}$.

Lemma 1: The lower bound of k_0 is given by $k_0\geq\max\left\{r_{max},\left\lceil\log_2\left(\frac{N}{c_{min}}\right)\right\rceil\right\}$.

Proof: The first term $k_0\geq r_{max}$ is simple, because the node with r_{max} missing pieces must need at least r_{max} cycles to receive the whole file since he can only get one file piece from others for each cycle. For the second term $k_0\geq\left\lceil\log_2\left(\frac{N}{c_{min}}\right)\right\rceil$, we focus on the distribution of one particular file piece. For every cycle, we try to best distribute this particular file piece. At cycle 0, there are c_{min} 1s along the column. At cycle 1, there will be $2\times c_{min}$ 1s along this column. At cycle 2, $2^2\times c_{min}$ 1s, and so on. At cycle k_0, there will be $2^{k_0}\times c_{min}$ 1s along this column, which should be greater than or equal to N. That is, $2^{k_0}\times c_{min}\geq N$, which gives $k_0\geq\log_2\left(\frac{N}{c_{min}}\right)$. As k_0 must be an integer, $k_0\geq\left\lceil\log_2\left(\frac{N}{c_{min}}\right)\right\rceil$. Hence, $k_0\geq\max\left\{r_{max},\left\lceil\log_2\left(\frac{N}{c_{min}}\right)\right\rceil\right\}$.

As an example, in the following problem instance, at least three cycles (not two) will be needed for complete distribution.

$$P=\begin{pmatrix}1&1&1&1&1&1&1&1\\0&1&1&1&1&1&1&1\\0&1&1&1&1&1&1&1\\0&1&1&1&1&1&1&1\\0&0&1&1&1&1&1&1\end{pmatrix}$$

This is because file piece 1 will need $k_0\geq\left\lceil\log_2\left(\frac{5}{1}\right)\right\rceil=3$ cycles, rather than $k_0\geq r_{max}=2$ cycles.

As mentioned before, our goal is to develop scheduling algorithms that minimize k_0. We shall use this lower bound to evaluate our algorithms through simulation results presented in later sections. Although this bound may not be tight for all problem instances, our simulation results indicate that it is tight for most of the cases.

4. Scheduling Algorithms

We now present three sets of transmission scheduling algorithms. They are *Rarest Piece First (RPF)*, *Most Demanding Node First (MDNF)*, and *Bipartite Matching (BPM)*. All of them run in polynomial time.

4.1. Rarest Piece First (RPF)

The *Rarest Piece First* algorithm is borrowed from the *Rarest Element First* algorithm employed in BitTorrent. In RPF, those file pieces that most peers do not have (rarest) are distributed first.

Definition 1: The rarity c_j of piece j is the number of

peers who have piece j. That is, $c_j = \sum_{i=1}^{N} P_{ij}$.

RPF aims at increasing the availability of different file pieces in the network, such that peers may still have some pieces that other peers want. In case the file is published by a single source who may just seed (remain available to contribute) the file for a short period of time, RPF also tries to distribute all pieces from the original source to different peers across the network as quickly as possible, so that the distribution can continue even if the original source leaves.

We classified two variations of the RPF algorithm based on the different orders for nodes to make decisions.

4.1.1. RPF – Node-Oriented. In the node-oriented variation, each peer chooses the rarest piece (with smallest c_j) to send out among those pieces it currently has. There is no preference on the choice of recipient; he only sends this piece to the one with the lowest row index who does not have this piece and has not been assigned to receive any piece yet. This process is done node by node (i.e. row by row in the possession matrix). The complexity of this algorithm is ***O(NM(N+logM))***, where ***N*** is the number of peers and ***M*** is the number of file pieces. For example,

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad P^1 = \begin{pmatrix} 1 & 0 & 1 & \underline{1} & 0 & 0 & 0 & 0 \\ 1 & 0 & \underline{1} & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \underline{0} & 0 & 1 & \underline{1} & 1 & 1 \\ 1 & 0 & 1 & 0 & \underline{1} & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & \underline{0} & 0 & 1 & 1 \end{pmatrix}$$

The schedule determined for P^0 is (arrows are put for easier understanding):

Node 1: send piece 3 to node 2
Node 2: send piece 4 to node 1
Node 3: send piece 5 to node 4
Node 4: send piece 6 to node 3

At P^0, node 1 is the first to make decision and it chooses piece 3 because among pieces 1 and 3 he currently has, piece 3 is the rarest (only two peers have it, while three peers have piece 1). Then he will send to node 2, since node 2 does not have piece 3 and has not been assigned to receive any piece yet. Node 2 is the next to select his choice. Similarly, node 2 chooses its rarest piece, piece 4, to send to node 1. This process continues node by node until no more transmissions can be scheduled. The resulting problem matrix at the next cycle, P^1, is also shown with those just transmitted pieces underlined. Note that in this case, only four transmissions can be scheduled (maximum is five), and this is the reason for its relatively poor performance when compared with other types of algorithms like MDNF, BPM that we will present later.

4.1.2. RPF – Piece-Oriented. Unlike the node-oriented variation, piece-oriented variation performs scheduling piece by piece (i.e. column by column) starting from the globally rarest piece first, then the second globally rarest piece, etc. For each piece (column), starts from the lowest row index node, if that node has this piece and has not yet been assigned to send any piece to others, assign him to send out this piece. There is again no preference on the choice of recipient; the peer just sends to the one with the lowest row index who does not have this piece and has not been assigned to receive any piece yet. This process continues until no more transmissions can be scheduled for this column; then we go to the next column until all columns are scheduled. Its complexity is ***O(M(N+logM))***. For example, in the same problem instance,

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \quad P^1 = \begin{pmatrix} 1 & 0 & 1 & \underline{1} & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & \underline{1} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & \underline{1} & 1 & 1 \\ 1 & \underline{1} & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & \underline{1} & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The schedule is determined in the following sequence:

Node 2: send piece 4 to node 1
Node 3: send piece 5 to node 2
Node 4: send piece 6 to node 3
Node 5: send piece 2 to node 4
Node 1: send piece 3 to node 5

At P^0, column 4 is selected for scheduling first since it is the globally rarest piece (only one peer has it, and its column number is the smallest). After node 2 sends this piece 4 to node 1, no more transmission can be made in piece 4, so we go to the next globally rarest piece, column 5. After column 5, then we schedule column 6. After column 6 is done, all rarest columns having only one possessing peer are done; then we go back to column 2 to schedule node 5 sending piece 2 to node 4. Node 3 cannot send again, as he has been assigned already. The process completes by scheduling the last column, column 3.

The performance of these two variations is quite similar as shown by simulation, with each outperforming the other in some cases. But in general, the RPF algorithms, which are derived from the *Rarest Element First* algorithm of BitTorrent, perform much worse than the following algorithms.

4.2. Most Demanding Node First (MDNF)

As indicated in Section 3.2, the number of cycles needed depends on two factors: how many pieces a peer needs and how rare a file piece is. To reduce the time for distribution, both factors have to be considered. RPF only considers the second factor but ignores the first one. The *Most Demanding Node First* algorithm takes care of the first factor by adding one additional criterion for choosing recipients and the performance improvement over RPF is significant with this simple enhancement.

Definition 2: The demand d_i of node i is the number of

un-received pieces for node i. That is, $d_i = \sum_{j=1}^{M} (1 - P_{ij})$.

We attach a demand d_i to every node and we prefer to send to the node with largest d_i. In case several nodes have the same demand, we just send to the node with the lowest row index.

Since MDNF is also based on RPF, we again have the same two types of variations.

4.2.1. MDNF – Node-Oriented. Similar to the *RPF – Node-Oriented* approach, we schedule the transmissions node by node; while choosing recipients we send to the one with the highest demand d_i that does not have the piece and has not been assigned to receive any piece yet. Its complexity is the same as *RPF – Node-Oriented*, i.e. ***O(NM(N+logM))***. For example,

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 6 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 6 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 4 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 4 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 5 \end{pmatrix} P^1 = \begin{pmatrix} 1 & 0 & 1 & \underline{1} & 0 & 0 & 0 & 0 \\ 1 & 0 & \underline{1} & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & \underline{1} & 1 & 1 \\ 1 & \underline{1} & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & \underline{1} & 0 & 1 & 1 \end{pmatrix}$$

The schedule is determined in this sequence:

Node 1: send piece 3 to node 2
Node 2: send piece 4 to node 1
Node 3: send piece 5 to node 5
Node 4: send piece 6 to node 3
Node 5: send piece 2 to node 4

The demands for each node are written at the right of P^0. At P^0, node 1 chooses its rarest piece, piece 3, to send out and chooses the most demanding node, node 2, to receive this piece. Similarly, node 2 sends piece 4 to node 1. Node 3 now sends piece 5 to node 5 (instead of node 4) because node 5 is more demanding than node 4, and so on for other nodes.

4.2.2. MDNF – Piece-Oriented. This is similar to the *RPF – Piece-Oriented* approach. Its complexity is ***O(M(logM+NlogN))***, and we present an example as follows.

$$P^0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 6 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 6 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 4 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 4 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 5 \end{pmatrix} P^1 = \begin{pmatrix} 1 & 0 & 1 & \underline{1} & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & \underline{1} & 0 & 0 & 0 \\ 0 & 1 & \underline{1} & 0 & 1 & 0 & 1 & 1 \\ 1 & \underline{1} & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & \underline{1} & 1 & 1 \end{pmatrix}$$

The schedule is determined in this sequence:

Node 2: send piece 4 to node 1
Node 3: send piece 5 to node 2
Node 4: send piece 6 to node 5
Node 5: send piece 2 to node 4
Node 1: send piece 3 to node 3

At P^0, the globally rarest piece, column 4, is selected first, so node 2 sends piece 4 to the most demanding node, node 1. Similarly, node 3 sends piece 5 to node 2. Then, node 4 sends piece 6 to node 5 because both node 1 and node 2 have been assigned and node 3 is less demanding than node 5. And it is similar for other nodes.

MDNF performs better than RPF in most cases but is still not optimal. A common characteristic that is shared between RPF and MDNF is that the maximum number of transmissions for each cycle cannot be achieved as in the example in Section 4.1.1. To fix the problem, we try to match as many sender and receiver pairs as possible in each cycle and the algorithm is described in the next section.

4.3. Bipartite Matching (BPM)

In this section, we present a novel bipartite matching graph model for finding transmission schedules which outperforms the above two sets of algorithms. We transform the problem instance to the well-known *maximum bipartite matching problem* so as to find as many sender and receiver pairs as possible in each cycle. Weights are added to the nodes to achieve better matching. We first describe how to transform the problem and then explain the algorithm in detail.

4.3.1. Problem Transformation. A *bipartite graph* is a graph $G = (V, E)$ such that there is a partition $V = L \cup R$, $L \cap R = \phi$, so that every edge connects a node from L with a node from R. A *matching* is a subset of the edges in E such that each vertex in V is incident to at most one edge in the matching. The maximum bipartite matching problem is to find a maximum matching, matching that consists the maximum number of edges, in a bipartite graph. The maximum bipartite matching problem can be reduced to the well-known *max-flow* problem, which can be solved by the Edmonds-Karp algorithm. To apply the algorithm, a flow network is constructed according to the bipartite graph. The flow network adds a *supersource* and a *supersink* to the bipartite graph. The supersource has an edge to each node in L and each node in R has an edge going to the supersink. Due to space limitation, we cannot describe the problem and the algorithm in detail. We refer interested readers to [11] for more formal discussion.

We now describe how to construct the flow network graph from a problem instance P.

Definition 3: The flow network graph from P is a directed graph $G = (V, E)$. $V = L \cup R \cup \{s, t\}$. L and R are the left and right batches of nodes having cardinality the same as the number of peers N, i.e., $|L| = |R| = N$. s, t are the supersource and supersink respectively. The edges consist of three sets, $E = \{(s,u) | u \in L\} \cup \{(v,t) | v \in R\} \cup C$, where $\{(s,u) | u \in L\}$, $\{(v,t) | v \in R\}$ are the sets of edges from the supersource to left batch nodes and right batch nodes to the supersink respectively.

$C=\{(u,v)\,|u\in L, v\in R, and\ \exists j(P_{uj}=1\wedge P_{vj}=0)\}$ are the edges from left to right batch nodes and depends on the possession matrix P_{ij}. There is an edge from u to v if u can be a sender to v, which means peer u has at least one file piece that peer v does not have. All edge capacities are one.

The following example illustrates the transformation process. For the following possession matrix P_{ij}, the flow network is as shown.

$$P=\begin{pmatrix}1&0&1&0&0&0&0&0\\1&0&0&1&0&0&0&0\\0&1&0&0&1&0&1&1\\1&0&1&0&0&1&0&1\\0&1&0&0&0&0&1&1\end{pmatrix}$$

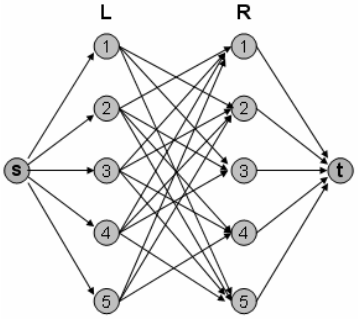
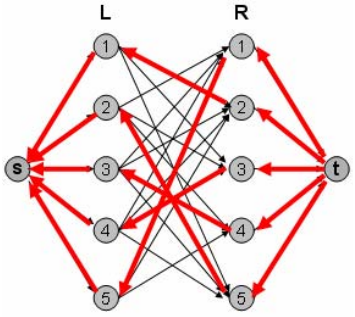


Figure 2. Possession matrix and its BPM graph

There are links from L1 to R2, R3 and R5, but not R4 because peer 1 can send piece 3 to peer 2, 3, and 5. However, peer 1 has nothing to send to peer 4 as peer 4 already has pieces 1 and 3 that peer 1 has. The arguments for other links are similar. There are $O(N^2)$ edges and the complexity for constructing the BPM graph is $O(N^2M)$.

To find the maximum matching $S\subseteq C$, we adopt the well-established Edmonds-Karp algorithm, which is a particular implementation of the general Ford-Fulkerson method [11]. It finds augmenting paths by using breath-first search from the supersource s to the supersink t. Nodes with smaller indices have higher preference in expanding states in breath-first search. Its complexity is $O(VE)=O(N^3)$. For the above example problem instance in Figure 2, we will have the following maximum matching S, with matched pairs highlighted.



$$P=\begin{pmatrix}1&0&1&0&0&0&0&0\\1&0&0&1&0&0&0&0\\0&1&0&0&1&0&1&1\\1&0&1&0&0&1&0&1\\0&1&0&0&0&0&1&1\end{pmatrix}$$

Figure 3. Maximally-matched BPM graph and the scheduled transmissions

After matching the pairs of sending and receiving nodes, we choose the rarest piece among these two matched nodes for transmission and this requires $O(M)$ time. The transmission schedule is thus,

Node 1: send piece 3 to node 2
Node 2: send piece 4 to node 5
Node 3: send piece 5 to node 4
Node 4: send piece 6 to node 3
Node 5: send piece 2 to node 1

However, although BPM always returns a schedule with the maximum number of transmissions for each cycle, the performance is not satisfactory, as it does not consider whether we can match more in subsequent cycles. For instance in the following counter example,
Using *BPM*, total 6 cycles are needed.

$$P^0=\begin{pmatrix}0&0&1&1&1\\1&0&0&1&1\\1&0&0&0&1\\1&1&0&0&1\\0&0&0&0&0\end{pmatrix}\ P^1=\begin{pmatrix}1&0&1&1&1\\1&0&1&1&1\\1&1&0&0&1\\1&1&0&0&1\\1&0&0&0&0\end{pmatrix}\cdots P^6=\begin{pmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{pmatrix}$$

Using *MDNF – Node-Oriented*, only 5 cycles are needed.

$$P^0=\begin{pmatrix}0&0&1&1&1\\1&0&0&1&1\\1&0&0&0&1\\1&1&0&0&1\\0&0&0&0&0\end{pmatrix}\ P^1=\begin{pmatrix}1&0&1&1&1\\1&1&0&1&1\\1&0&0&1&1\\1&1&0&0&1\\0&0&1&0&0\end{pmatrix}\cdots P^5=\begin{pmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{pmatrix}$$

The key point is that at P^0, the maximally-matched schedule (4 transmissions) by *BPM* blocks peer 5 from contributing anything at P^1, as all other peers have piece 1 already. The better one is *MDNF – Node-Oriented*; which lets peer 5 to also contribute something at P^1, thus shortening the whole distribution time by one cycle.

4.3.2. BPM – Simple-weighted. The problem of BPM is that it does not take care of the rarity of file pieces and the demands of nodes as in RPF and MDNF in matching. To find a better matching, we put weights in the nodes so as to give priorities to some nodes during the matching process. For example, if we give a higher weight to peer 5 in the above example, peer 5 will be matched and will not be idle.

We put weights on nodes on both sides and these weights reflect the demands of the peers and the rarity of the file pieces they possess. Definition 2 defines the *demand* d_i of a peer in Section 4.2. We now define how to measure the rarity of the file pieces a peer possesses.

Definition 4: The rarity possession index γ_i of peer i is the sum of number of **0s** in other peers for those pieces that peer i has. That is, $\gamma_i=\sum_{a=1}^{N}\sum_{b=1}^{M}(B_{ab}(i))$ where B_{ab} is an $N\times M$ matrix and $B_{ab}(i)=\begin{cases}1 & if\ \ (a\neq i)\wedge(P_{ib}=1)\wedge(P_{ab}=0)\\ 0 & if\ \ otherwise\end{cases}$.

For example, for P in Figure 2, $\gamma_1 = 2 + 3$ (number of zeros in column 1 plus number of zeros in column 3).

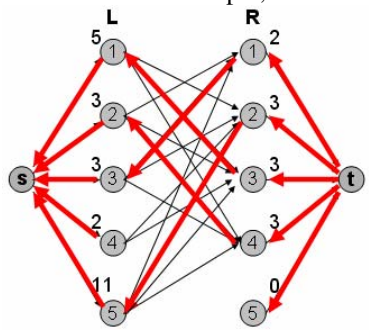
Our *BPM – Simple-weighted* algorithm works as follows:

1. Construct the flow network graph according to Definition 3.
2. Find the rarity possession indices of all peers and

assign the values as weights on the nodes in L accordingly.

3. Find the demands of all peers and assign the values as weights on the nodes in R accordingly.
4. Employ the Edmonds-Karp algorithm to find the weighted maximal matching.
5. For each matching, identify the rarest piece to be sent.

Consider an example,



$$P=\begin{pmatrix}0&0&1&1&1\\0&1&1&0&0\\0&1&0&1&0\\0&0&1&1&0\\1&1&1&1&1\end{pmatrix}$$

Figure 4. BPM – Simple-weighted graph and the scheduled transmissions

The numbers besides the right-side nodes R are the demands d_i, and the numbers besides the left-side nodes L are the rarity possession indices γ_i. For example, L1 has $\gamma_1 = 5$ because for pieces 3, 4, 5 that peer 1 has, there are a total of five 0s along columns 3, 4, and 5. By preferring paths with largest γ_i first, we ensure those peers who have rare pieces can send first. On the other hand, the algorithm tends to select a node with higher demand to be a receiver, thus making sure the most demanding nodes can get file pieces.

Simulation results show that *BPM – Simple-weighted* performs better than MDNF and RPF. However, there are still a few cases that it cannot achieve the optimal as shown below.

Using *BPM – Simple-weighted*, total 5 cycles are needed.

$$P^0=\begin{pmatrix}1&0&1&1&0&1&0&0\\0&0&0&1&1&0&1&1\\0&0&1&0&1&1&0&1\\1&1&0&0&1&0&1&0\\0&0&1&0&1&1&1&0\\1&1&1&1&0&0&0&0\\1&1&0&0&1&0&1&0\\1&0&0&1&0&1&1&0\end{pmatrix}\quad P^1=\begin{pmatrix}1&0&1&1&0&1&0&1\\0&1&0&1&1&0&1&1\\0&0&1&1&1&1&0&1\\1&1&1&0&1&0&1&0\\0&1&1&0&1&1&1&0\\1&1&1&0&1&1&0&0\\1&1&0&1&0&1&1&0\\1&1&0&1&0&1&1&0\end{pmatrix}\quad \ldots P^5 = \mathbf{1}$$

An optimal schedule requires only 4 cycles.

$$P^0=\begin{pmatrix}1&0&1&1&0&1&0&0\\0&0&0&1&1&0&1&1\\0&0&1&0&1&1&0&1\\1&1&0&0&1&0&1&0\\0&0&1&0&1&1&1&0\\1&1&1&1&0&0&0&0\\1&1&0&0&1&0&1&0\\1&0&0&1&0&1&1&0\end{pmatrix}\quad P^1=\begin{pmatrix}1&0&1&1&0&1&0&1\\0&0&1&1&1&0&1&1\\0&1&1&0&1&1&0&1\\1&1&0&0&1&0&1&1\\0&1&1&0&1&1&1&0\\1&1&1&1&0&1&0&0\\1&1&1&0&1&0&1&0\\1&1&0&1&0&1&1&0\end{pmatrix}\quad \ldots P^4 = \mathbf{1}$$

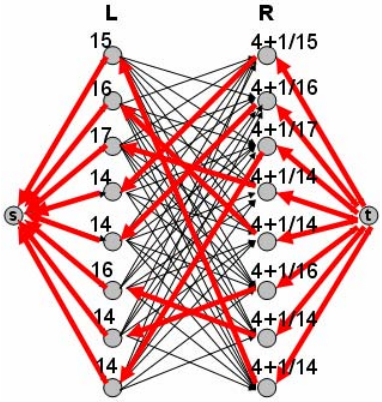
The reason is that P^0 is very special in that all demands $d_i=4$ are the same, and so the weights on R become useless. In addition, the rarity possession indices γ_i only ensure those peers with larger γ_i can be matched with someone else, but do not imply those rarest pieces can actually be sent to those in need. In the above example at P^1, *BPM – Simple-weighted* matches peer 1 with 2, peer 2 with 3, and peer 3 with 1, which by no ways the rarest piece 7 can be sent out to others.

4.3.3. BPM – Enhanced-weighted. Due to the above problem, we further enhance *BPM – Simple-weighted* by adding the *rarity demand index* δ_i (defined below) to the demand d_i to make up the total weight for the right-side nodes R. That is, right-side weights = $d_i + \delta_i$.

Definition 5: The rarity demand index δ_i of peer i is the reciprocal of the sum of number of 1s in other peers for those pieces that peer i does not have. That is, $\delta_i = \frac{1}{\sum_{a=1}^{N}\sum_{b=1}^{M}(D_{ab}(i))}$ where D_{ab} is an $N \times M$ matrix and

$$D_{ab}(i)=\begin{cases}1 & if \quad (a\neq i)\wedge(P_{ib}=0)\wedge(P_{ab}=1)\\ 0 & if \quad otherwise\end{cases}.$$

A final example to illustrate the solution of the previous problem,



$$P=\begin{pmatrix}1&0&1&1&0&1&0&0\\0&0&0&1&1&0&1&1\\0&0&1&0&1&1&0&1\\1&1&0&0&1&1&0&0\\0&0&1&0&1&1&1&0\\1&1&1&1&0&0&0&0\\1&1&0&0&1&0&1&0\\1&0&0&1&0&1&1&0\end{pmatrix}$$

Figure 5. BPM – Enhanced-weighted graph and the scheduled transmissions

The right-side weights are in the form $d_i + \delta_i$. For example, R1 has $\delta_1 = 1/15$ because for pieces 2, 5, 7, 8 that peer 1 does not have, there are a total of fifteen 1s along columns 2, 5, 7 and 8. By adding the rarity demand indices δ_i to the right-side weights, those peers having the rarest pieces will be matched with those who really need these rarest pieces first. Note that $\delta_i = 1/d_i$ is not necessarily true in general. The example above is just a special case only.

BPM – Enhanced-weighted finds optimal schedules for all cases we tested in simulations. Therefore, we conjecture that *BPM – Enhanced-weighted* is an optimal scheduling algorithm for solving this simplified P2P collaborative file distribution problem.

5. Simulation Results

We randomly generate the problem instances (with each individual matrix element independently generated) and employ various algorithms presented in the previous section to find transmission schedules. We note the number of cycles needed by using that particular algorithm and compare it with the lower bound we developed in Section 3.2. We use the sub-optimal ratio as the performance measure.

Definition 6: The sub-optimal ratio is the number of cases that the particular algorithm cannot return the lower bound number of cycles needed over the total number of cases simulated. That is, $sub\text{–}optimal\ ratio = \frac{num\ of\ sub\text{–}optimal\ cases}{total\ num\ of\ cases}$.

Below is the graph for the sub-optimal ratios for all the above mentioned algorithms over varying problem size with ***N=M***. We simulated 100,000 cases for each simulation point.

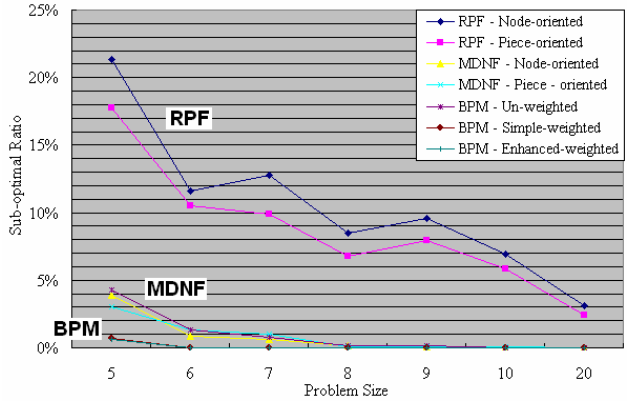


Figure 6. Sub-optimal ratios of various scheduling algorithms (all) over varying problem size

For a particular set of algorithms (i.e. RPF, MDNF, BPM), the performance of different variants are similar. The performance of MDNF is always better than RPF, while the performance of BPM is always better than MDNF (with the exception of *BPM – Un-weighted*). We find that for small problem sizes (e.g. 5x5, 6x6), the RPF algorithms perform very poorly; with about 10-20% cases returning sub-optimal solutions. With increasing problem size, the sub-optimal ratio decreases. For RPF and MDNF, the sub-optimal ratios are generally higher (e.g. small peak at 7x7, 9x9) for odd problem sizes. It is because RPF and MDNF generally match peers with one another (e.g. peer 1 send to peer 3, and peer 3 send back to peer 1), thus making one odd peer idle, degrading the performance. This odd number problem does not happen in BPM. In Figure 7, we magnify Figure 6 by only showing the sub-optimal ratios for MDNF and BPM.

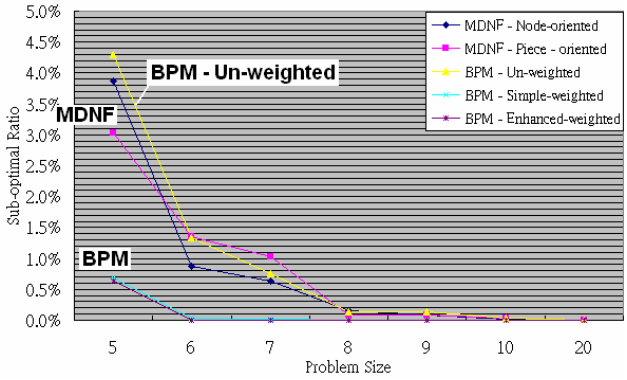


Figure 7. Sub-optimal ratios of various scheduling algorithms (MDNF+BPM) over varying problem size

With the exception of *BPM – Un-weighted*, BPM algorithms outperform all other types of algorithms. The reason for the unsatisfactory result of *BPM – Un-weighted* has been discussed in Section 4.3.1. The small sub-optimal ratios of weighted BPM for small problem sizes (eg. 5x5, 6x6, 7x7) is due to the loose lower bound estimation for special problem instances like this.

$$P^0=\begin{pmatrix}1&1&1&1&1\\0&0&1&1&1\\0&0&1&1&1\\0&0&1&1&1\\0&0&1&1&1\end{pmatrix}\ P^1=\begin{pmatrix}1&1&1&1&1\\1&0&1&1&1\\0&0&1&1&1\\0&0&1&1&1\\0&0&1&1&1\end{pmatrix}\ P^2=\begin{pmatrix}1&1&1&1&1\\1&0&1&1&1\\0&1&1&1&1\\1&0&1&1&1\\0&0&1&1&1\end{pmatrix}\ \ldots P^4=\mathbf{1}$$

The optimum is 4 cycles, which is higher than the lower bound of 3 cycles predicted. We check all those "sub-optimal" cases of *BPM – Enhanced-weighted* and find that all of them belong to the above special class. Therefore, we find no cases that *BPM – Enhanced-weighted* returns sub-optimal solution and thus conjecture that it is an optimal scheduling algorithm.

We further perform simulations for varying file size (***M***) with fixed peer size (***N***), and the case for varying peer size with fixed file size.

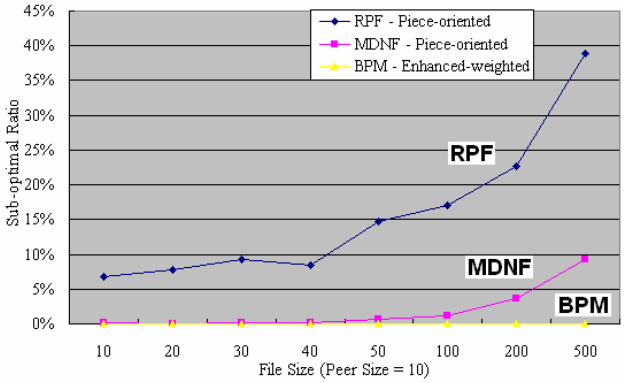


Figure 8. Sub-optimal ratios of various scheduling algorithms over varying file size (peer size = 10)

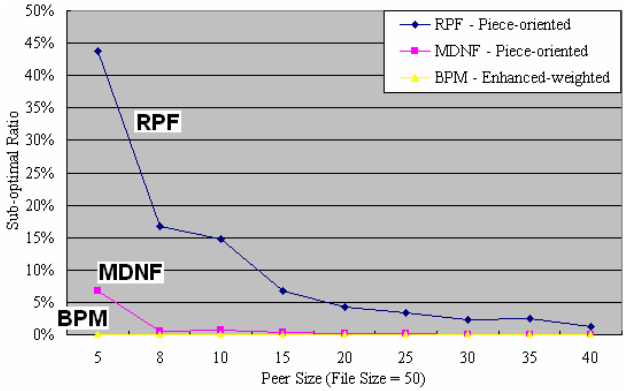


Figure 9. Sub-optimal ratios of various scheduling algorithms over varying peer size (file size = 50)

From Figure 8, the sub-optimal ratios of RPF and MDNF increase with increasing file sizes, while from Figure 9, the sub-optimal ratios of RPF and MDNF decrease with increasing peer sizes. The rate of increase with increasing file size is smaller than the rate of decrease with increasing peer size, thus making the sub-optimal ratios drop with increasing problem size as shown in Figure 6 and 7. In all cases, *BPM – Enhanced-weighted* returns the optimal solution.

6. Future Work

In this paper, we have investigated the simplified problem of P2P file distribution scheduling with symmetric bandwidth and homogeneous network assumptions. We shall study the case for asymmetric bandwidth, which is common in the Internet as domestic users usually connect using the Asymmetric Digital Subscriber Line (ADSL) technology where uploading bandwidth is smaller than downloading bandwidth. We shall also study the case when the network is heterogeneous with asynchronous transmission time for different pairs of nodes.

7. Conclusion

Peer-to-Peer file sharing applications have become immensely popular in the Internet, but previous research seldom investigates the data distribution problem which should be the core of any file sharing applications. We formally define the collaborative file distribution problem with the possession matrix and transmission matrix formulation and suggest several types of algorithms (RPF, MDNF and BPM) for solving the scheduling problem of deciding who send which file pieces to whom. In particular, our novel Bipartite Matching model outperforms all other algorithms and can return the optimal solution for all cases as shown by simulations. Therefore, we conclude that the *BPM – Enhanced-weighted* algorithm is a promising algorithm for practical deployment as the core scheduling algorithm in P2P file sharing applications.

8. Acknowledgement

This research is supported in part by the Areas of Excellence Scheme, established by the University Grants Committee, Hong Kong Special Administrative Region, China, Project No. AoE 99-01.

9. References

[1] P. Rodriguez, and E.W. Biersack, "Dynamic Parallel Access to Replicated Content in the Internet," *IEEE/ACM Transactions on Networking*, Vol. 10, No. 4, pp. 455-465, August 2002

[2] The Official BitTorrent Website, *http://www.bittorrent.com/*

[3] The Official Gnutella Website, *http://www.gnutella.com/*

[4] The Official Kazaa Website, *http://www.kazaa.com/*

[5] The Official Napster Website, *http://www.napster.com/*

[6] B. Cohen, "Incentive Build Robustness in BitTorrent," *http://www.bittorrent.com/bittorrentecon.pdf*, May 2003

[7] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna, "Efficient Collective Communication in Distributed Heterogeneous Systems," *Proc. International Conf. on Distributed Computing Systems*, pp. 15-24, June 1999

[8] S. Khuller, and Y.A. Kim, "On Broadcasting in Heterogeneous Networks," *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pp. 1011-1020, January 2004

[9] M. Banikazemi, V. Moorthy, and D.K. Panda, "Efficient Collective Communication on Heterogeneous Networks of Workstations," *International Conf. on Parallel Processing*, pp. 460-467, August 1998

[10] P. Liu, "Broadcast Scheduling Optimization for Heterogeneous Cluster Systems," *Journal of Algorithms*, Vol. 42, No. 1, pp. 135-152, January 2002

[11] T. H. Cormen, C.E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2nd edition, 2001